Computer Networking Final Project

第九組

組員：

機械四 鄭澄遠

機械四 龔柏翰

題目：直播軟體-雍塞控制與影音串流

大 綱 & 目錄：

程 式 與 功 能 簡 介

在這個高速網路的時代，遊戲直播越來越熱門，現今最大遊戲直播平台 twitch 光在台灣每個月不重複訪客數就到達了 450 萬人次，電子競技重要賽事時最高同時在線人數甚至達到幾十萬；除了遊戲直播之外，網紅直播也很熱門，M17 直播集團最近也進軍日本。直播產業在全世界都在以急遽的速度成長，所以我們也想做出一個類似的系統。

於是我們設計出一個簡單的直播系統，可以同時使用 UDP 與 TCP 來傳送不同需求的資料，達到最佳的傳輸效果與最適當影像品質。其中包括了：

1. 影像傳輸

2. 多人連線

3. 用戶畫質調整

等功能。如此讓直播主可以透過一台攝影機讓很多的觀眾可以順暢的在同時間觀賞自己的節目。

程式使用 python 撰寫，因為其套件最為豐富，而且我們也比較熟悉。我們使用的 py 套件有：cv2、numpy、threading、socket 等。

實 作 語 言 與 套 件

我們使用 python 來撰寫程式。

其中比較重要的套件有：

1. cv2：用來獲取影像資訊。包括：
   - ◆ VideoCapture：用來獲取影像
   - ◆ set：設定獲取影像大小
   - ◆ imencode：可接獲取的影像編碼成 JPG
2. socket：用來開啟 TCP 與 UDP
   - ◆ socket.SOCK_DGRAM：可開啟 UDP 連線
   - ◆ socket.SOCK_STREAM：可開啟 TCP 連線
3. threading：平行程式執行
   - ◆ Thread：可同時執行兩種成程序。因應封包需求，我們需要同時使用 TCP 與 UDP，所以我們需要用到這個函式。
   - ◆ join：加入新的程序
   - ◆ start：開始執行
   - ◆ current_thread：獲取現在 thread 的資訊
4. numpy：做一些簡單的矩陣運算

函 式 簡 介

我們在主程式(server)自行設計了兩個函式，主要用於整理影像並傳輸：

1. video_streaming

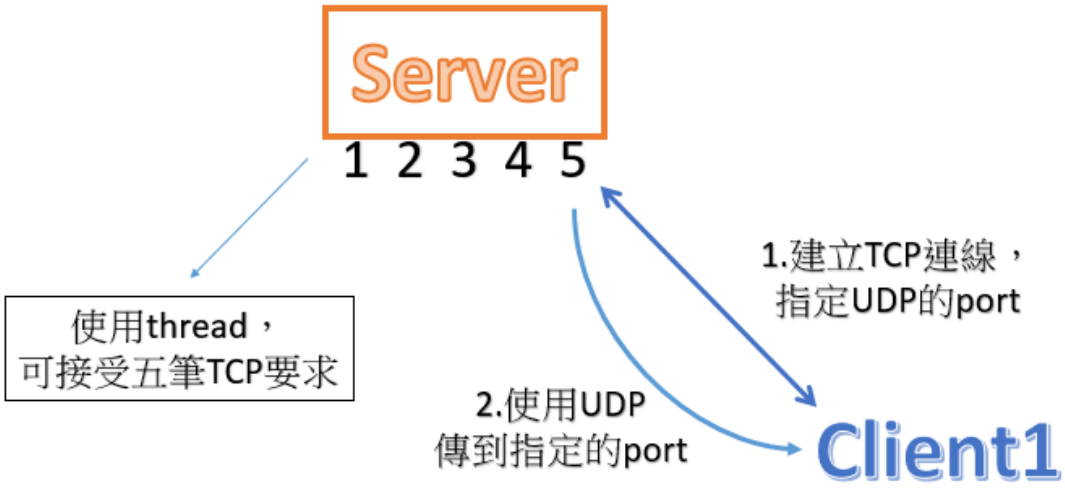
    - 輸入指定品質與封包大小
    - 讀取攝影機內容
    - 用 cv2.imencode 將影像依照指定參數壓縮並編碼成 JPG
    - 將 JPG 做成一維矩陣，並轉成字串
    - 開始傳輸至 client

2. do_stream

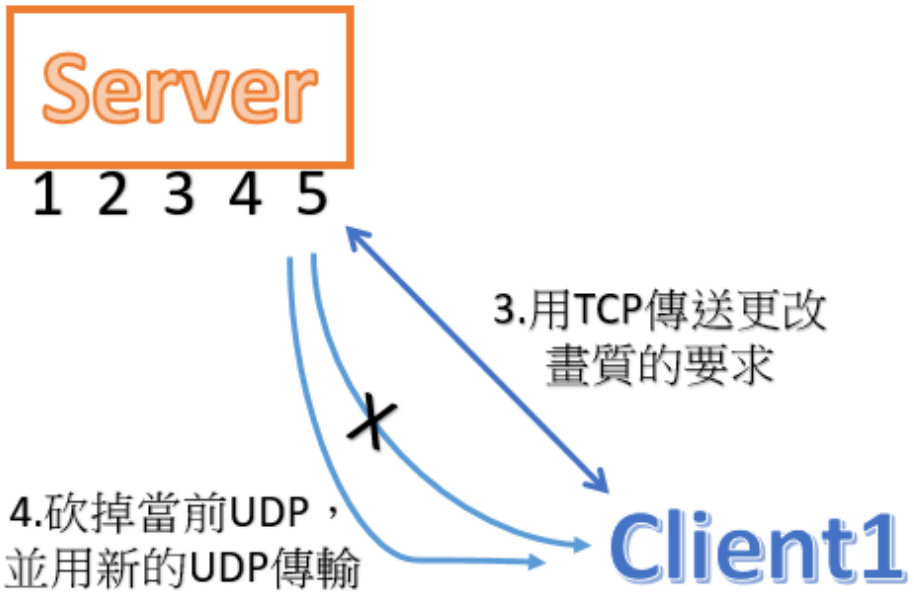
    - 告知 vedio_streaming 其所需要的參數(封包大小、品質等)
    - 使用 vedio_streaming 連續不斷的傳送封包
    - 針對用戶端的畫質要求做出反應

Server 主 程 式 流 程 簡 介

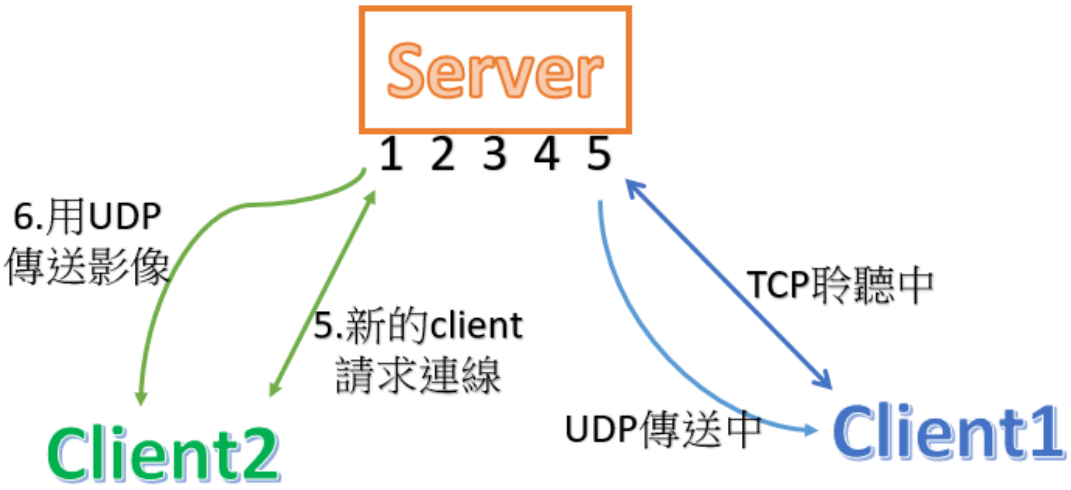
1. 建立 TCP 與 UDP 連線，開啟一個 thread，開始聆聽(listen)
2. 攝影機初始化
3. 接收 client 端的 TCP 連線，並連線，獲得 client 端資訊(IP、port 等)
4. 連線完成後，開始使用 UDP 傳輸影像至指定的 port，使用函式 do_stream 與 vedio_streaming 傳輸
5. 等待 client 的畫質參數要求，並根據要求變更影像畫質
6. 當偵測到第二個 client 的連線需求時，開啟第二個 thread，新增兩個程序，一個是 UDP，一個是 TCP，供新的 client 使用。
7. 圖示：



▲ client1 連線到 server



▲ client1 要求畫質變更



▲ client2 連線到 server

client 程 式 介 紹

1. 開啟 UDP 與 TCP 連線

2. 用 TCP 跟 server 連線，並將 UDP 的 port 告訴 server

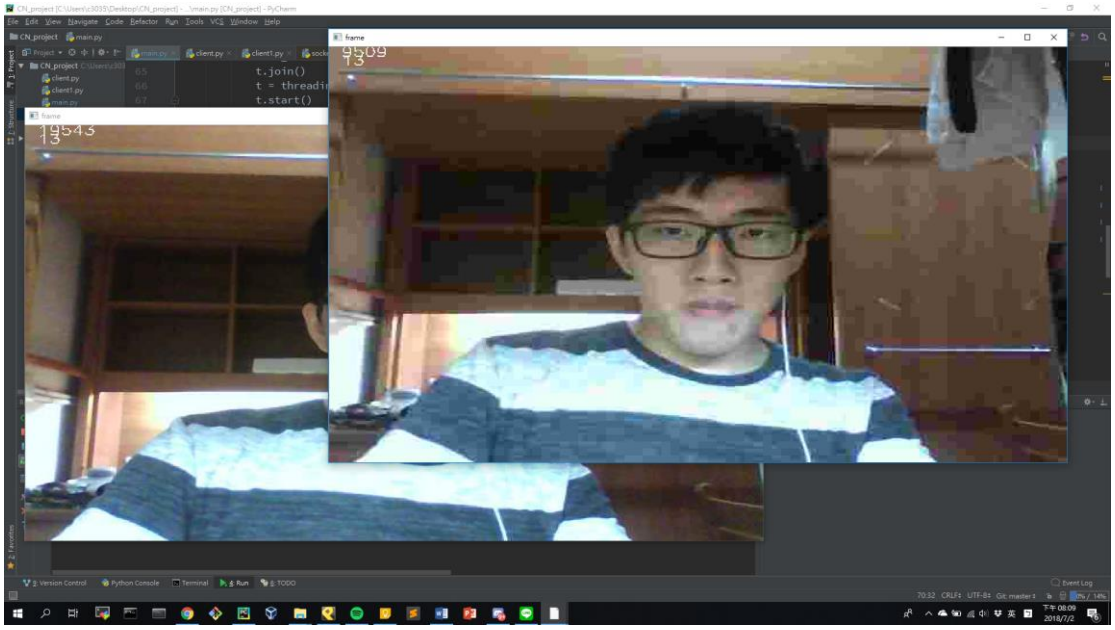
3. 使用指定的參數(畫質、封包大小)跟 server 請求影像

4. 使用剛剛指定的 port 接收影像並解碼

5. 展示影像，同時展示當前的封包大小與每秒收到的封包數。

6. 等待用戶的變更畫質需求並且用 TCP 傳送新的參數告訴 Server

成 果 展 示



▲左邊視窗：封包大小 19453，每秒 13 個；右邊視窗：封包大小 9500 每秒 13 個。可以看出左邊的 client 視窗畫質比較高。

▲最高畫質，封包大小 19500，每秒 13 個 。可以清晰地顯示影像。

過 程 中 的 困 難 與 解 決 辦 法

1. Q：影像跟用戶需求的參數不適合同時使用 TCP 或 UDP，影像部分因為傳輸量大，又可允許微量封包遺失，所以選用 UDP；用戶需求的參數部分傳輸量少，且不可容忍封包遺失，所以使用 TCP。於是我們需要同時使用兩種不同的傳輸層協定。

   A：使用 threading，並行 TCP 與 UDP 程序，來讓影像使用 UDP 的同時，有一筆 TCP 的連線可以讓 Server 與 Client 互相了解對方的需求並做出回應。

2. Q：畫質調整問題。

   A：每個 client 的網路雍塞程度不同，所以我們有必要降低畫質來讓用戶可以順暢的觀賞節目。所以我們讓 server 的 TCP 連線始終都在聆聽 client 的畫質需求。當 client 端用戶覺得畫質需要變更時，會用 TCP 發送一組字元，當 server 偵測到這筆字元時，會根據要求開啟新的 UDP，並把舊的 UDP 刪除。如此才能讓畫質不同的影像無縫接軌的保持順暢的播放。

3. Q：多人同時連線的問題。

   A：使用 threading，當 server 偵測到新的一筆連線要求時，其

會開啟一筆新的 threading 以供這個新的 client 使用，並用此 threading 開啟新的 TCP 與 UDP 來服務這個新 client。因其獨立於舊的 client 的 threading，所以可以使用不同的畫質來傳輸，也可以辨識某份畫質要求是從哪個 client 傳送而來的，並且做出相對應的參數調整。

4. Q：影像壓縮與傳送問題

A：首先我們使用 JPG 來傳輸，而 JPG 本身就有壓縮了，壓縮後的大小會跟當下畫面的色彩複雜度有關，但通常都為幾十 KB，這個大小算是易於使用網路傳輸。再來傳輸的部分，因為他只接受 16 進位的傳輸方式，所以我們需先將訊號矩陣轉換才能傳輸，於是我們先用 tostring() 將檔案轉成字串(16 進位)，再做傳輸。

5. Q：影像分割傳送問題

A：我們一開始先將影像分割再傳輸，不過這個方法在封包遺失時，會導致畫面某些格子遺漏(一條線)，也就是整個畫面會有一小塊是毀損的，更勝者，其可能無法以剛剛的順序重組，導致整畫面崩壞。所以我們在測試了很多種分割數之後，決定不要分割畫面，直接整個畫面傳送過去，如有遺失就整個畫面挑過，原因

為我們一秒鐘大概都有 10 幾張，如果有封包的遺失，看就觀看者角度而言，其實影響尚屬輕微，肉眼也不易察覺，只有在遺失比較多張時，會產生 lag 的情況；而如果是使用分割畫面，因為是 UDP 的關係，只要分割的包裹中有一個毀損，他播出來的畫面也是損毀的。與其播出損毀的畫面，我們認為乾脆將此張拿掉，就觀看者的角度而言，反而覺得影響不是那麼嚴重。

未 來 展 望

1. 增加安全性：可以對封包進行加密與解密，增加傳輸安全性。
2. 可以增加聊天系統:讓直播主可以收到觀眾的回饋，並可以用此與觀眾互動。